Intel® XML Software Suite 1.2 for Java* Environments

User's Guide

Copyright © 2007–2008 Intel Corporation

All Rights Reserved

Document Number: 317575-005US

Revision: 1.5

World Wide Web: http://www.intel.com

Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

Copyright (C) 2007-2008, Intel Corporation. All rights reserved.

Revision History

Document Number	Revision Number	Description	Revision Date
317576	002	Intel® XML Software Suite 1.0 for Java* Environments	December 2007
317576	004	Intel® XML Software Suite 1.1 for Java* Environments	August 2008
317576	005	Intel® XML Software Suite 1.2 for Java* Environments	October 2008

Table Of Contents

1 Overview 1
About This Document 1
Product Overview 1
Supported Environments 3
Major Features 4
Technical Support 7

2 Getting Started 8
Quick Start 8
Getting Version Information 10
Configuring the Intel® XML Software Suite 11
Sample Applications 13
Working with the Intel® XML Software Suite 17

3 Using the Intel® XML Parsing Accelerator 28
Parsing Data in SAX Mode 28
Parsing Data in DOM Mode 29
Parsing and Writing Data in StAX Mode 32
Enabling Data Validation 38
Enabling DTD Validation 39

4 Using the Intel® XML Schema Accelerator 41
Validating Data 41
Configuring Validation 43

5 Using the Intel® XSLT Accelerator 45
Performing the XSL Transformation 45
Customizing the Intel® XSLT Accelerator 46
Object Types 47
XSLT Extensions 48

6 Using the Intel® XPath Accelerator 56
Performing the XPath Evaluation 56
Resolving External Resources 58

7 Troubleshooting 61

Appendix A: Acronyms and Definitions 64

Appendix B: Compatibility Modes: Support for various Interpretations of XSL Conditions 66

Appendix C: References 73

Index 75

1 Overview

About This Document

This document describes the Intel® XML Software Suite, with an overview of its major features and a high-level view of the API and intended usage. With this document, you will be able to:

- start using the Intel XML Software Suite
- run the samples supplied with the package
- employ advanced functionality of the libraries making up the suite
- troubleshoot product-related issues.

The document is targeted at developers that need to perform XML data processing in their applications. The document assumes knowledge of XML and Java* programming skills.

NOTE

When viewing this guide in PDF format, your Adobe* Reader* viewer might generate hyperlinks from URL-formatted text of the guide. To disable this option, modify your preferences to disable automatic hyperlink generation.

Product Overview

The Intel® XML Software Suite for Java* environments is an XML processing library that you can use to parse, process and serialize the XML data used in your application. The library implements the JAXP [9] standard for XML processing in Java* so that it can be easily used as an alternative to your normal JAXP provider, be that a default implementation installed in your Java* runtime environment or a third-party implementation. You can do an easy drop-in replacement and use this product without changing your application.

The Intel XML Software Suite provides a rich, industrial-strength implementation of the JAXP API, which ensures exceptional performance. Now you can obtain the best that the Intel® architecture can offer and take advantage of such new features as the character and string processing instructions in the Intel® Streaming SIMD Extensions 4.2 (Intel® SSE 4.2), and scalability across multiple cores.

The Intel XML Software Suite covers all major stages of XML data processing, with each stage handled by a separate component, as follows:

- The Intel® XML Parsing Accelerator parses XML data to the Simple API for XML (SAX) API or a W3C Document Object Model (DOM).

    o Simple API for XML (SAX) model represents data as a sequence of events.

    o Streaming API for XML (StAX) model provides a bidirectional API for reading and writing. It exposes a simple iterator-based API and the underlying streaming of events.

    o Document Object Model (DOM) model represents data as a node tree.

  After the document is parsed, the data can be further processed by other components in the library.

- The Intel® XML Schema Accelerator validates an XML document against a specified W3C XML schema [7] document. XML Schemas define a set of validation constraints which can be used to ensure that the XML data has the correct structure and values for your application to use.

- The Intel® XSLT Accelerator transforms an XML document into another XML, XHTML or text document by applying a W3C XSLT 1.0 stylesheet. The stylesheet describes how to transform the input document to the output document format.

- The Intel® XPath Accelerator navigates XML data by evaluating an XML path (W3C XPath) [8] expression over an XML document or a DOM tree. The expressions return references to parts of the document or values extracted from the document for your application to use.

The Intel® XML Software Suite can be plugged into different environments to accelerate XML processing, for example:

- Enterprise service busses. They introduce intermediaries between the client and the back-end business service which may parse the XML data passed in, validate that it conforms to a schema, perform an XSLT transformation to change and augment its format, and extract routing information using XPath.

- Application servers. They may provide web services stacks, perform data mashups, and much more. In web services stacks, messages may be parsed, validated and transformed.

The Intel® XML Software Suite offers various interfaces to the underlying XML data representation, providing functionality for SAX, StAX or DOM parsing, schema or DTD validation, XSLT transformation, and XPath expression evaluation. It is recommended to use the suite for all the functions required, instead of combining functional elements with other JAXP implementations. For example, instead of generating a DOM with Apache Xerces Java Parser and performing an XSLT transformation on the result with the Intel XML Software Suite, use the Intel libraries for both these functions. This can result in significantly greater efficiency gains.

Supported Environments

The Intel® XML Software Suite is supported on the following Java* environments:

- Sun Java 2 Platform, Standard Edition* (J2SE) 5.0
- Sun Java Platform, Standard Edition* (Java SE) 6
- Oracle JRockit* 5.0
- Oracle JRockit* 6.0
- IBM JDK*, Version 5.0 (Linux only)
- IBM SDK* for Java 6 (Linux only)

The Intel XML Software Suite is supported on the following Windows* and Linux* operating systems on IA-32 and Intel® 64 architectures, as well as other x86 architectures:

- The Linux* OS:
  - Red Hat* Enterprise Linux* Server and Advanced Platform 5.0
  - Red Hat* Enterprise Linux* AS 4.0 - 2.6 kernel
  - Red Hat* Enterprise Linux* ES 4.0 - 2.4 kernel
  - Red Hat* Enterprise Linux* AS 3.0 - 2.4 kernel
  - Red Hat* Enterprise Linux* ES 3.0 - 2.4 kernel
  - SUSE* Linux* Enterprise Server 10 - 2.6 kernel
  - SUSE* Linux* Enterprise Server 9 - 2.6 kernel
- The Windows* OS:
  - Windows Server 2003 Standard Edition*
  - Windows Server 2003 DataCenter Edition*
  - Windows Server 2003 Enterprise Edition*
  - Windows Server 2008*
  - Windows XP*
  - Windows Vista*

The Intel XML Software Suite achieves the best performance on the Intel® architectures, by using features of the Intel® platforms and compiler tuning. However, new software algorithms enable large performance gains on other platforms as well. If you need support for the platforms not listed above, send a request to xmlinfo@intel.com.

The Intel® XML Software Suite for Java* Environments is now available on HP-UX* OS 11iV2 supporting the Itanium®-based platforms. For additional product details, please see The Intel® XML Software Suite for Java* Environments on HP-UX* OS.

Major Features

All libraries of the Intel® XML Software Suite possess the following key features:

- *High Performance*: The libraries making up the suite demonstrate high performance by using an underlying native XML processing core. In the product, major processing is done inside the core which interacts with client application via the standard Java* native interface (JNI*). This architecture is transparent to the application.
- *Portable and Platform-Specific Performance:* the libraries making up the suite can use the available features of the underlying hardware platform. On the Intel® Core TM i7 with Intel® Streaming SIMD Extensions 4.2 (Intel® SSE 4.2) that accelerate string and character processing, or Intel® platforms supporting multiple cores, you can get significant performance gains without changing the software configuration. The Intel SSE 4.2 use two forms of parallelism:
    - performing operations on multiple 8- or 16-byte characters at a time, using the Intel Architecture's 256-bit XMM registers.
    - performing several operations concurrently on the array of characters.
- *Conformance and Compatibility*: The libraries of the Intel XML Software Suite conform to the following standards defining XML data processing:
    - W3C* XML 1.0 [1]
    - W3C Namespaces in XML 1.0 [2] except "Qualified Names in Declarations"

- W3C DOM level 2.0 core and partially DOM level 3 core [4], with the limitations and specifics defined in the product *Release Notes*
- SAX 2.0 Core and Extensions [5] with specifics defined in the product *Release Notes*
- W3C XML Conformance Test Suite [6] except the XML 1.1 and namespaces in XML 1.1 cases
- Java* API for XML Processing, JAXP [9], a de-facto standard for XML processing in the Java environments; the current release supports JAXP 1.3 and JAXP 1.4

In addition to these, the components making up the suite possess a number of specific features, as described in Table 1.

Table 1. Component-Specific Features

Component	Conformance	Key characteristics	Supported file size
Intel® XML Parsing Accelerator		• Operates with data in the format of data streams, DOM trees or SAX events • Can enable validation of documents with the Intel® XML Schema Accelerator before passing data to the application • Can run in the namespace-awareness mode • Performs DTD and data validation • Provides significant	32-bit systems: SAX mode up to 20GB, DOM mode up to 300MB; 64-bit systems: SAX mode up to 20GB, DOM mode up to 600MB

Component	Conformance	Key characteristics	Supported file size
		performance gains on the Intel® Core™ i7 with Intel SSE 4.2.	
Intel® XML Schema Accelerator	W3C* XML Schema 1.0 [7] except PSVI (post-schema-validation infoset)	• Provides significant performance gains on the Intel® Core™ i7 with Intel SSE 4.2.	Up to and above 1GB
Intel® XSLT Accelerator	W3C XSLT 1.0 [3]; namespace declaration formats specific for the Apache* Xalan* XSLTC [11] processor; OASIS* conformance test suite [6] for the Apache* Xalan-Java* processor	• Operates with data in the format of data streams, DOM trees or SAX events • Supports XSLT extensions, including EXSLT extension functions [13] to allow applications to extend XSLT functionality • Supports parallel transformation of an XML document with multiple threads	Up to and above 1GB
Intel® XPath Accelerator	W3C XML Path Language (XPath) 1.0 [8]		Up to and above 1GB

Technical Support

To receive technical support for the Intel® XML Software Suite, register for an Intel Premier Support account at the Registration Center.

To register for an account, visit the Intel® Registration Center web site. If you have forgotten your password, send a request to: quadsupport@mailbox.intel.com. Please do not send your technical issue to this e-mail address.

Additional information and support are available from the Intel® Software Network-XML User Forum and XML Software Blog.

To submit an issue via the Intel® Premier Support web site, do the following:

1. Make sure that Java* and JavaScript* are enabled in your browser.
2. Go to http://premier.intel.com and log in.
3. Submit your issue:
    a. Click **Submit Issues** in the left column.
    b. Select the product type from the **Product Type** drop-down menu. For the Intel® Software Products, choose **Development Environment** (tools, SDV, EAP) as the product type.
    c. Select the product for which you want to submit the issue from the **Product Name** drop-down menu.
    d. Enter a headline-type description of the problem or question in the **Issue Title** text box.
    e. Enter a detailed question, or a detailed problem description in the **Question** text box.
    f. Select **Request Type** from the drop-down menu.
    g. To complete the submission process, click **Submit Issue** or **Submit Issue / Upload Files**, if you need to attach a file, for example, a test case or a screenshot. You will get an issue number that enables you to track your issue.

2 Getting Started

Quick Start

This section of the User's Guide explains how to quickly set up the library for use in your environment.

After installing the Intel® XML Software Suite as described in the *Installation Guide,* do the following:

1. Check that the content of the installation directory matches the description in the README file.

   If discrepancies occur, consult the Troubleshooting section for recommendations on resolving the issue.

2. On Linux*, enable Java* signal processing [14], for example:

   - **For IA-32 architectures:**

     ```
     export LD_PRELOAD=<JAVA_HOME>/jre/lib/i386/libjsig.so
     (KSH)
     setenv LD_PRELOAD <JAVA_HOME>/jre/lib/i386/libjsig.so
     (CSH)
     ```

   - **For Intel® 64 architectures:**

     ```
     export LD_PRELOAD=<JAVA_HOME>/jre/lib/amd64/libjsig.so
     setenv LD_PRELOAD (KSH)
     <JAVA_HOME>/jre/lib/amd64/libjsig.so (CSH)
     ```

3. **Option 1:** Check that your environment is set up correctly. This step is usually done automatically at the installation stage. The environment variables should be set as shown in Table 2.

   Option 2: Modify the way you start your application, following the steps below. It can speed up the class loading process and provide additional flexibility:

   1. Specify JAXP [9] factories on your Java* command line for the components you wish to use by modifying the system properties as shown in Table 3. For example:

```
java –Djavax.xml.parsers.SAXParserFactory=
com.intel.xml.sax.SAXParserFactoryImpl
```

2. Set the run-time parameters to specify the path to the Intel XML Software Suite native library.

- **For IA-32 architectures:**
  `java -Djava.library.path=<`**`install_dir`**`>/bin/ia32`
- **For Intel® 64 architectures:**
  `java -Djava.library.path=<`**`install_dir`**`>/bin/intel64`

Table 2. Environment Settings

OS/ Platform	Variable Name	Linux* OS	Windows* OS
IA-32	CLASSPATH	**<install_dir>**/lib/intel-xss.jar	**<install_dir>**\Lib\intel-xss.jar
	PATH	N/A	**<install_dir>**\Bin\ia32
	LD_LIBRARY_PATH	**<install_dir>**/bin/ia32	N/A
Intel® 64	CLASSPATH	**<install_dir>**/lib/intel-xss.jar	**<install_dir>**\Lib\intel-xss.jar
	PATH	N/A	**<install_dir>**\Bin\intel64
	LD_LIBRARY_PATH	**<install_dir>**/bin/intel64	N/A

Table 3. Configuring JVM to use the Intel XML Software Suite

Component	VM Property and Value
SAX Parser	`-Djavax.xml.parsers.SAXParserFactory= com.intel.xml.sax.SAXParserFactoryImpl`
DOM Parser	`-Djavax.xml.parsers.DocumentBuilderFactory= com.intel.xml.dom.impl.ESDocumentBuilderFactory`
XML Reader	`-Dorg.xml.sax.driver=`

Component	VM Property and Value
	`com.intel.xml.sax.XMLReaderImpl`
XSL Transformer	`-Djavax.xml.transform.TransformerFactory= com.intel.xml.transform.TransformerFactoryImpl`
Schema Validator	`-Djavax.xml.validation.SchemaFactory: http://www.w3.org/2001/XMLSchema= com.intel.xml.validation.impl.SchemaFactoryImpl`
XPath	`-Djavax.xml.xpath.XPathFactory: http://java.sun.com/jaxp/xpath/dom= com.intel.xml.xpath.impl.XPathFactoryImpl`
StAX components	`-Djavax.xml.stream.XMLInputFactory= com.intel.xml.stream.XMLInputFactoryImpl -Djavax.xml.stream.XMLOutputFactory= com.intel.xml.stream.XMLOutputFactoryBase -Djavax.xml.stream.XMLEventFactory= com.intel.xml.stream.XMLEventFactoryImpl`

Similar settings need to be set to enable Intel XML Software Suite in Eclipse* IDE.

Getting Version Information

The Intel® XML Software Suite enables you to get the library version information using the `GetVersion() API`.

The Version Query samples demonstrate how to extract the version information and view it from the command-line interface. These samples are located in the `<install_dir>/examples/common/VersionQuery` folder.

The example code below shows how to run a Version Query Sample:

```
import com.intel.xml.XMLToolkit;
import com.intel.xml.XMLToolkit.Version;
public class InlinedVersionQuery {
       //Prints the version Info of Intel XML Software Suite to
the standard output
             public static void main(String argv[])
      {
//Getting Version Information as a String
```

```
          System.out.println(XMLToolkit.getVersionAsString());
//Getting Version Information Object
          Version version = XMLToolkit.getVersion();
//Printing version information – Major: 1, Minor:2, Patch:0
          System.out.println("Major: " +
version.getMajorVersion());
          System.out.println("Minor: " +
version.getMinorVersion());
          System.out.println("Patch: " +
version.getPatchVersion());
// Show build number of last build before release
          System.out.println("BuildNumber: " +
version.getBuildNumber());
          System.out.println("Platform: " +
version.getPlatform());
```

See also:

Version Query Samples

Configuring the Intel® XML Software Suite

The Intel® XML Software Suite is distributed with the `config.xml` configuration file in the **<install_dir>**`/conf` directory that enables you to control some product behaviors. For example, you may want to increase or decrease the memory threshold to speed up data processing or to reduce memory consumption accordingly.

After successfully installing the Intel XML Software Suite, you can delete, re-create, or modify the configuration file as needed. With the configuration file, you can:

- Adjust the *threshold of memory usage* to restrict the amount of system virtual memory consumed by the Intel XML Software Suite. The default value is 512MB.
- Configure *parallel XSLT transformation* to allow the Intel® XSLT Accelerator to manipulate an XML document in multiple threads. The value of this option must be an integer. The default value is 0, so the feature is disabled; a larger value enables parallel transformation and sets the number of maximum working threads. However, the Intel XSLT Accelerator can further adjust the maximum number of threads; for instance, if you set this option to 128 threads, the Intel XSLT

Accelerator may choose to create only 4 threads for optimal performance.

- Configure `DocumentBuilderFactory` to ignore all unsupported features and attributes. By default, `DocumentBuilderFactory` throws exceptions for unsupported features and attributes. To ignore the unsupported features and attributes, set the throw-unknown-feature value as false.

```
<component name ="Intel XSLT Accelerator">
<property name="namespace-prefix" value="false" />
</component>
```

The settings in the configuration file can apply to all or specific components. They have the following format:

```
<property name="name_of_property" value="current_value" />
```

Component-specific settings are inside respective component tags, for example:

```
<component name ="Intel XSLT Accelerator">
      <property name="maximum memory" value="512m" />
</component>
```

On startup, the Intel XML Software Suite loads and parses `config.xml` to get the settings. If no correct value is found or if an error occurs, the product uses the default built-in value. If the library `libintel-xss-j.so` (on the Linux* OS) or `intel-xss-j.dll` (on the Windows* OS) is not in the folder where it was originally installed, the Intel XML Software Suite searches for `config.xml` at `../../conf/` relative to the folder where the library is located.

Configuring Components

In addition to centralized configuration settings via an external file, you can adjust the operation of individual Intel XML Software Suite components programmatically, specifically:

- The Intel® XML Parsing Accelerator:
    - Enable input data validation mode, see Enabling Data Validation
    - Enable DTD validation mode, see Enabling DTD Validation
- The Intel® XML Schema Accelerator
    - Set a custom error handler or an external resource resolver, see Configuring Validation

- The Intel® XSLT Accelerator
    - Set output properties and add parameters to the XSL stylesheet, see Configuring the Intel® XSLT Accelerator
    - Add your own extension functions, see User-Defined Extension Functions
- The Intel® XPath Accelerator
    - Set external resource resolvers for namespace contexts, XPath variables and functions, see Resolving External Resources

Sample Applications

he distribution of the Intel® XML Software Suite includes many source code examples that show how you can use the product with your application. These examples are packaged with source files, `makefiles`, and sample XML input documents. You can run the examples over the provided XML sample files or with your own files.

To run an example, navigate to the corresponding directory in the **<install_dir>**`/examples/<`**component**`>` directory, and run the code. For example, to run the DOM writer example for the Intel® XML Parsing Accelerator, do the following:

1. To compile the source code, type:

```
javac Writer.java
```

For the InlinedVersionQuery example, include the intel-xss.jar into the CLASSPATH, as follows:

```
javac –cp <install_dir>/lib/intel-xss.jar
InlinedVersionQuery.java
```

2. To run the sample, type:

```
java Writer personal.xml
```

The samples are divided into several groups by the component they use.

For the Intel® XML Parsing Accelerator

The Intel® XML Parsing Accelerator can process data in SAX and DOM modes. The examples for this component are grouped into `dom/` and `sax/` subdirectories respectively.

dom/Writer	Traverses a DOM tree to print a document that has been parsed.

dom/Counter	Traverses a DOM tree to get information about the whole document.
dom/DOMAddLines	Adds lines to the DOM Node.
dom/GetElementsByTagName	Uses the `Document#getElementsByTagName()` method to quickly and easily locate elements by the tag name.
sax/Counter	Registers a SAX2 `ContentHandler` and receives the callbacks in order to print information about the document.
sax/DocumentTracer	Provides a complete trace of SAX2 events for the files parsed. This is useful for making sure that a SAX parser implementation communicates all the information in the document to the SAX handlers.
sax/Writer	Registers a SAX2 `ContentHandler` and receives callbacks to print the document that is parsed.
StAX/StreamReader/Counter	Iterates with a StAX `StreamReader` to count the number of events in the document that is parsed.
StAX/StreamReader/Tracer	Iterates with a StAX `StreamReader` to print the document.
StAX/EventReader/Counter	Iterates with a StAX `EventReader` to count the number of events in the document that is parsed.
StAX/EventReader/Tracer	Iterates with a StAX `EventReader` to print the document that is parsed.
StAX/EventWriter/SimpleEventWriter.java	Writes a series of XML Events through

	the `XMLEventWriter`.
`StAX/StreamWriter/SimpleStreamWriter.java`	Writes a sample XML document through the `XMLStreamWriter`.
`StAX/StreamWriter/RepairNamespaceWriter.java`	Demonstrates the namespace repairing feature of the StAX `Writer`.

For the Intel® XML Schema Accelerator

`jaxp/` **`SourceValidator`**	Consists of the **`ValidatingParser`** and **`Validator`** samples, which demonstrate validation of XML documents in the stream, SAX and DOM modes against the XML Schema documents.

For the Intel® XSLT Accelerator

`Transform`	Performs a stream-to-stream transformation for the specified `xml and xsl files`.
`ExtensionFunction`	Shows how to make use of extension functions defined in the Java* language during XSLT transformation. Specifically, the stylesheet in this example invokes Java methods defined in `java.lang.Integer` and `java.land.String` classes. For more information on using extensions with the Intel® XSLT Accelerator, see XSLT Extensions.
`Redirect`	Uses the Xalan-specific `redirect` extension to divide transformation output and write parts of it to specified files instead of dumping the whole output into a single `javax.xml.transform.Result` object. For more details on the example, consult the Apache* Xalan-Java* website [10].
`Apache* Xalan* Samples`	
	This directory holds selected samples from the Apache* Xalan-Java* web site [10] that show ease of adapting your Apache Xalan-Java based application to the Intel® XSLT Accelerator. Each example is located in a separate subdirectory. With this release, the following examples are

	supplied:
SimpleTransform	Uses the `xsl` stylesheet to convert an `xml` source file and print the output.
trax	Demonstrates the use of the Transformation API for XML processing (TrAX).
SAX2SAX	Explicitly sets the SAX `XMLReader` and SAX `ContentHandler` for processing the `xsl` stylesheet and the `xml` input files, and producing the output.
Chain transformations	Feed output of one transformation as input into another: • **Pipe** illustrates a forwarding model where each `Transformer` object directs its output into the following transformer. • **UseXMLFilters** illustrates a backward model where each `Transformer` object is an extension of the SAX `XMLFilter` interface and sets each preceding filter as the parent of the following filter in the chain.
TransformThread	Spawns multiple threads, with each thread running two transformations on two different XML files.
UseStylesheetPI	Reads the stylesheet processing instruction in the XML source file to select the stylesheet for the transformation. For more information, see Configuring the Intel® XSLT Accelerator.
UseStylesheetParam	Takes a stylesheet parameter, reads the `xml` source file and the `xsl` stylesheet, and performs the transformation. The stylesheet parameter appears as a text node in the output. For more information on these parameters, see Configuring the Intel® XSLT Accelerator.

In addition to example subdirectories, the `ApacheXalanSamples` directory contains the supporting file `serializer.jar`. Most samples need `serializer.jar` only at

compile time, but `Pipe`, SAX2SAX and `UseXMLFilters` samples need it both at compile time and at run time.

For the Intel® XPath Accelerator

`ApplyXPathJAXP`	Evaluates an XPath expression over an input XML source.
`XPathVariable`	Evaluates an XPath expression with variables over an XML source.
`XPathExternalFunction`	Evaluates an XPath expression referring to an external function over a pre-built DOM tree.

Version Query Samples

`InlinedVersionQuery`	Gets the version info of the Intel XML Software Suite by importing corresponding classes.
`StandaloneVersionQuery`	Gets the version info of the Intel XML Software Suite by invoking corresponding methods with reflect mechanism.

Multi-component Samples

`BookStore`	Demonstrates how to use Intel DOM parsing, validation, XPath, and XSLT functionality in a sample workflow of online book purchasing.
`SoapMessage`	Shows how to use Intel XPath and validation capabilities on a SOAP message.

Working with the Intel® XML Software Suite

Before working with the Intel® XML Software Suite, set up your environment as specified in Quick Start.

Working in Eclipse* IDE

You can now develop your XML and SOA applications from Eclipse* IDE using the Intel® XML Software Suite for Java* Environments. This section describes how to integrate the Intel XML Software Suite into the Eclipse 3.3 IDE as an external library for your application or as a resource for your plug-ins.

Note

Before starting the integration, make sure that Eclipse IDE and the Intel XML Software Suite are successfully installed on your machine. See the *Installation Guide* for instructions.

Adding the Intel XML Software Suite to a Project

Complete these steps to add a new .jar library to your Eclipse project:

1. Create a new project or open an existing one.
2. Right-click your project and select **Properties**.
3. Go to **Java Build Path** > **Libraries** tab.
4. Click **Add External JARs...** and navigate to the Intel XML Software Suite installation directory. On the Linux* OS: `<install_dir>/lib`. On the Windows* OS: `<install_dir>/Lib`.
5. Select `intel-xss.jar`, click **Open**, and **OK**.
6. In the dialog box **JARs and class folders on the build path:**, click the `intel-xss.jar` name to open the drop-down list. See Figure 1.
7. In the list of options, double-click **Native library location:**, click **External Folder...**, and navigate to the location of the Intel XML Software Suite. For Linux*:
   - On 32-bit systems: `<install_dir>/bin/ia32/intel-xss-j.so`
   - On 64-bit systems: `<install_dir>/bin/intel64/intel-xss-j.so`
8. Click **OK**, then **OK** in the **Properties** window.

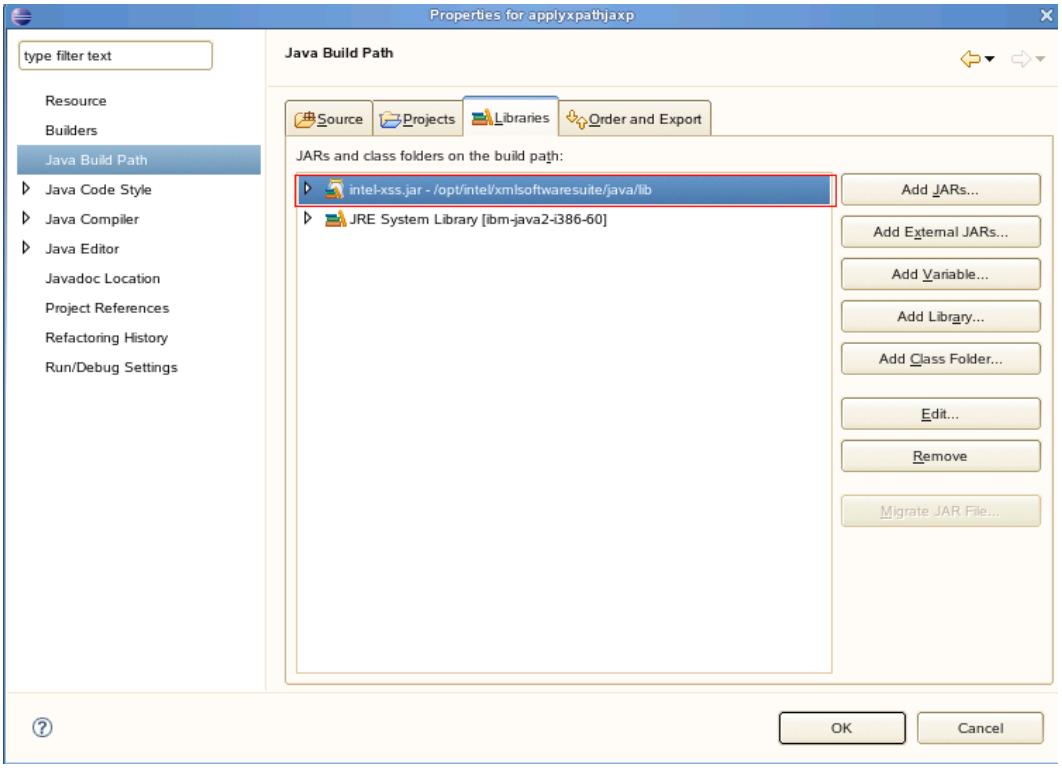

Figure 1. Specifying Location of the Intel XML Software Suite library in Eclipse IDE for Linux*.

The Intel XML Software Suite library is now available in your project, and you can use it in your Java application to perform XML data processing.

Using the Intel XML Software Suite in Eclipse IDE

To use the Intel XML Software Suite as the default XML data processing engine, move `intel-xss.jar` to the top of the external libraries list, which is located in the **Order and Export** tab of the project properties, see Figure 2.

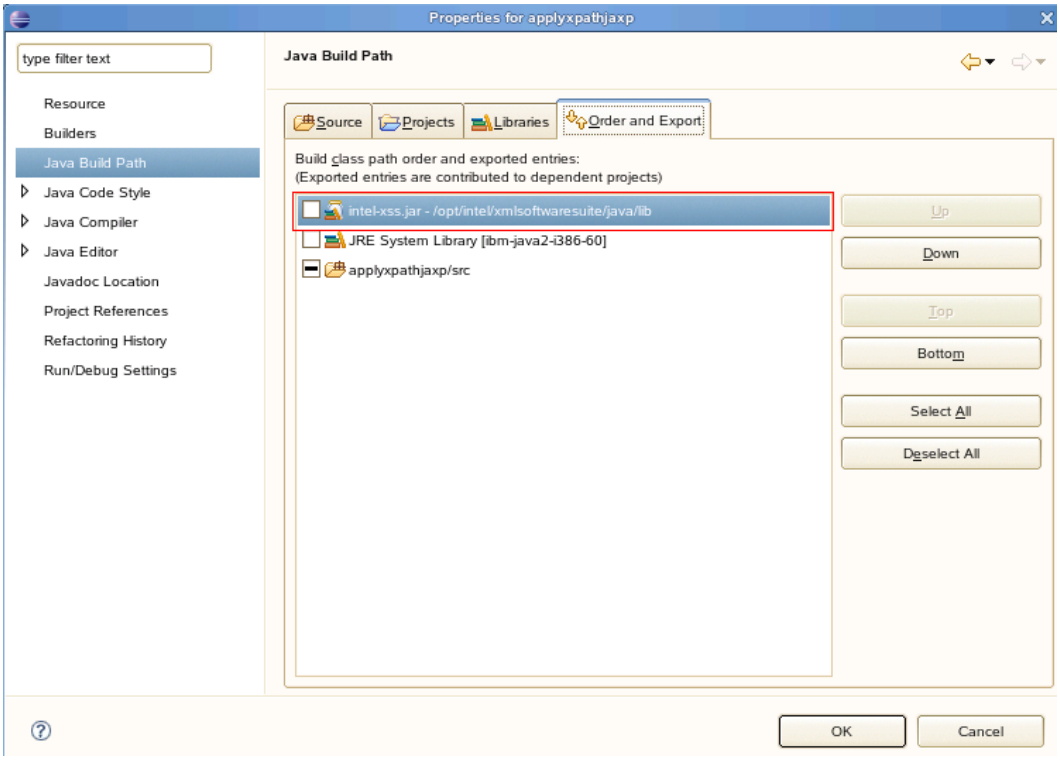

Figure 2. Enabling the Intel XML Software Suite as the processing engine

Alternatively, you may keep your current processing engine as the default library, and configure the JVM to use the Intel XML Software Suite capabilities in specific cases. You can do this in the configuration (running or debugging), by adding new properties to **Arguments** > **VM augments**; see Table 3 for a list of available Intel XML Software Suite key classes for different modes of XML data processing. Figure 3 demonstrates setting up a debugging configuration with XSLT Transformation performed by the Intel XML Software Suite.

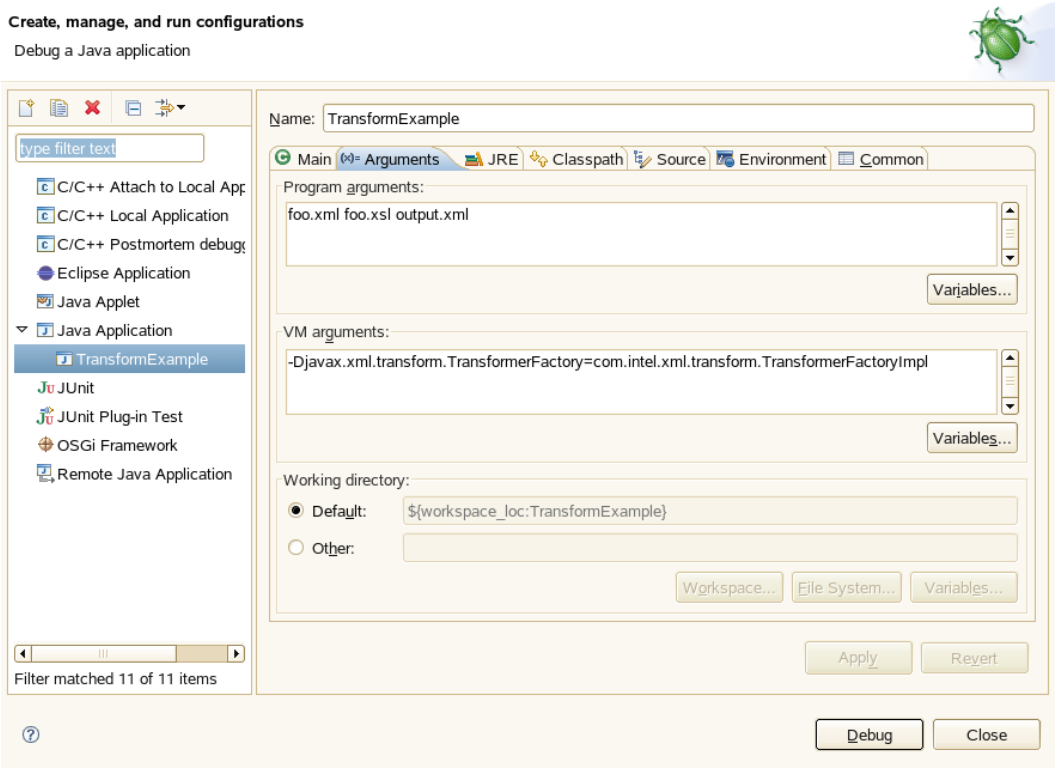

Figure 3. Setting up a Configuration to do XSLT Transformation with the Intel XML Software Suite

Integrating with Application Servers

Java* EE (J2EE) Application Servers are the key components of the enterprise computing infrastructure, especially in service-oriented architecture (SOA) environments. The Intel® XML Software Suite can be integrated with application servers to process XML data. You can replace the XSLT portion of the default XML processing engine with the Intel XML Software Suite completely or can use it in specific instances.

Application servers use shell scripts to set Java runtime parameters and environment variables. These scripts may ignore the CLASSPATH variable set by the Intel XML Software Suite installer.

This section shows how to integrate the Intel XML Software Suite into the application server to make the application server scripts load its libraries appropriately.

Apache* Tomcat* Servlet/JSP Container 5.5/6.0

To enable the Intel XML Software Suite with Apache Tomcat, do one of the following:

Option 1: Replace the default XML processor server-wide:

1. a) For the Windows* OS, put `intel-xss.jar` into `TOMCAT_HOME\common\endorsed` (for Tomcat 6.0, the path is `TOMCAT_HOME\lib`) or `JAVA_HOME\jre\lib\ext`
   b) For the Linux* OS, put `intel-xss.jar` into `TOMCAT_HOME/common/endorsed` (for Tomcat 6.0, the path is `TOMCAT_HOME/lib`) or `JAVA_HOME/jre/lib/ext`

2. a) For the Windows* OS, put `intel-xss-j.dll` into `JAVA_HOME\jre\bin`

   b) For the Linux* 32-bit systems, put `libintel-xss-j.so` into `JAVA_HOME/jre/lib/i386`

   c) For the Linux* 64-bit systems, put `libintel-xss-j.so` into `JAVA_HOME/jre/lib/amd64`

Option 2: Replace the default XML processor application-wide:

1. a) For the Windows* OS, put `intel-xss.jar` into `WEB_APP_ROOT\WEB-INF\lib`
   b) For the Linux* OS, put `intel-xss.jar` into `WEB_APP_ROOT/WEB-INF/lib`

2. a) For Windows*, put `intel-xss-j.dll` into `JAVA_HOME\jre\bin`

   b) For the Linux* 32-bit systems, put `libintel-xss-j.so` into `JAVA_HOME/jre/lib/i386`
   c) For the Linux* 64-bit systems, put `libintel-xss-j.so` into `JAVA_HOME/jre/lib/amd64`

Oracle* WebLogic* Server 9.X/10.X

Enabling for Applications

To enable the Intel XML Software Suite for applications, replace the default XML processor application-wide:

1. Copy the `intel-xss.jar` file to the web application library:

   a) For Windows*, put `intel-xss.jar` into `%WEB_APP_ROOT%\WEB-INF\lib`

   b) For Linux*, put `intel-xss.jar` into `$WEB_APP_ROOT/WEB-INF/lib`

2. Copy the native library of the Intel XML Software Suite to the java runtime library path of the web application:

   a) For Windows*, put `intel-xss-j.dll` into `%APP_SERVER_JDK_HOME%\jre\bin`

   b) For the Linux* 32-bit OS, put `intel-xss-j.so` into `$APP_SERVER_JDK_HOME/jre/lib/i386`

c) For the Linux* 64-bit OS, put `intel-xss-j.so` into `$APP_SERVER_JDK_HOME/jre/lib/amd64j`

NOTE

`WEB_APP_ROOT` is the directory of the user application; `APP_SERVER_JDK_HOME` specifies the jdk home the web server container uses to run the application.

Enabling Server-wide

To enable the Intel XML Software Suite as the default processor for DOM, SAX, XML Schema, XSLT and XPath handling, you need to configure the Java system properties and CLASSPATH settings.

The Intel StAX factories must be omitted from the system properties of the Oracle* WebLogic* Server configuration due to a known limitation in the StAX component of the Intel XML Parsing Accelerator.

To configure the system properties and CLASSPATH, do the following:

For the Windows* OS:

- ```
  SET JAVA_OPTIONS=%JAVA_OPTIONS%
  -Djava.library.path=<install_path>
  \XMLSoftwareSuite\Java\1.2\Bin\ia32;
  <BEA_HOME>\wlserver_10.0 <or
  weblogic90>\server\native\win\32;
  -Djavax.xml.parsers.SAXParserFactory=
  com.intel.xml.sax.SAXParserFactoryImpl
  -Djavax.xml.parsers.DocumentBuilderFactory=
  com.intel.xml.dom.impl.ESDocumentBuilderFactory
  -Dorg.xml.sax.driver= com.intel.xml.sax.XMLReaderImpl
  -Djavax.xml.transform.TransformerFactory=
  com.intel.xml.transform.TransformerFactoryImpl
  -Djavax.xml.validation.SchemaFactory:http:
  //www.w3.org/2001/XMLSchema=com.intel.xml.validation.impl.S
  chemaFactoryImpl -Djavax.xml.xpath.XPathFactory:http:
  //java.sun.com/jaxp/xpath/dom=
  com.intel.xml.xpath.impl.XPathFactoryImpl
  ```

- ```
  SET CLASSPATH=%CLASSPATH%;
  <install_path>\XMLSoftwareSuite\Java\1.2\Lib\intel-xss.jar;
  ```

For the Linux* OS:

- ```
  JAVA_OPTIONS="${JAVA_OPTIONS}-Djava.library.path=
  <install_path>/XMLSoftwareSuite/Java/1.2/bin/ia32:
  ```
  ```

```
<BEA_HOME>/wlserver_10.0 <or
weblogic90>/server/native/linux/i686;
-Djavax.xml.parsers.SAXParserFactory=
com.intel.xml.sax.SAXParserFactoryImpl
-Djavax.xml.parsers.DocumentBuilderFactory=
com.intel.xml.dom.impl.ESDocumentBuilderFactory
-Dorg.xml.sax.driver= com.intel.xml.sax.XMLReaderImpl
-Djavax.xml.transform.TransformerFactory=
com.intel.xml.transform.TransformerFactoryImpl
-Djavax.xml.validation.SchemaFactory:http:
//www.w3.org/2001/XMLSchema
=com.intel.xml.validation.impl.SchemaFactoryImpl
-Djavax.xml.xpath.XPathFactory:http:
//java.sun.com/jaxp/xpath/dom=com.intel.xml.xpath.impl.XPat
hFactoryImpl" export JAVA_OPTIONS
```

- `CLASSPATH="${CLASSPATH}:<install_path>/XMLSoftwareSuite/Java/1.2/lib/intel-xss.jar"`
  `export CLASSPATH`

NOTE

To prevent the Oracle* WebLogic* Server start-up failures, append the Intel XML Software Suite at the end of the Java CLASSPATH.

Tip

For your convenience, you can add the commands listed above to the start-up command script executed by the WebLogic Server during the initialization. For example:

On Windows:

`<WLS_install_path>\user_project\domains\<DOMAIN_NAME>\bin\setDomainEnv.cmd`

On Linux:

`<WLS_install_path>/user_project/domains/<DOMAIN_NAME>/bin/setDomainEnv.sh`

IBM* WebSphere* Application Server V6.1

IBM* WebSphere* is designed to set up, operate, and integrate e-business applications across multiple computing platforms using Java-based technology. Now Intel XML Software Suite can be successfully integrated with IBM WebSphere to accelerate the XML transactions. After installing IBM WebSphere, follow these steps to replace the default XML processor server-wide:

1. Copy `intel-xss.jar` to the Java extension library directory for the Application Server:
    - For the Windows* OS: put `intel-xss.jar` into `WEBSPHERE_ HOME\APPLICATION_SERVER\java\jre\lib\ext`
    - For the Linux* OS: `put intel-xss.jar` into `WEBSPHERE_ HOME/APPLICATION_SERVER/java/jre/lib/ext`
2. Copy the Intel native library to the Java runtime library path for the Application Server:
    - For the Windows* OS: put `intel-xss-j.dll` into `WEBSPHERE_ HOME\APPLICATION_SERVER\java\jre\bin`
    - For the Linux* 32-bit systems: put `intel-xss-j.so` into `WEBSPHERE_ HOME/APPLICATION_SERVER/java/jre/lib/i386`
    - For the Linux* 64-bit systems: put `intel-xss-j.so` into `WEBSPHERE_ HOME/APPLICATION_SERVER/java/jre/lib/amd64`

NOTE

Integration has only been tested with WebSphere version 6.1 and JDK 1.5.

If you have a problem, start up the server with the trace option on and refer to the following log files for more information:

- Windows:
    - Command: `startServer.bat <serverName> -trace`
    - Log Location: `APPLICATION_SERVER\profiles\AppSrv01\logs\<serverName>;`
- Linux:
    - Command: `./startServer <serverName> -trace`
    - Log Location: `APPLICATION_SERVER/profiles/AppSrv01/logs/<serverName>;`

See also:

Replacing the Default XML Parser

Replacing the Default XML Parser

Some web service applications, such as Oracle AquaLogic Service Bus*, are built on top of Apache XMLBeans*, which provides a number of facilities for XML processing. Older XMLBeans implementations only worked with the Piccolo* default SAX parser. This section explains how to configure XMLBeans to use the Intel® XML Parsing Accelerator.

First create an xbean.jar file using the patch `patch_alt_parser.txt`:

Note

You may need to customize these steps for your environment.

1. Download the patch `patch_alt_parser.txt` from https://issues.apache.org/jira/browse/XMLBEANS-378. This patch enables you to use other SAX parsers.

2. Download the source code from http://xmlbeans.apache.org/sourceAndBinaries/.
   For example, for version 2.3.0 use the command
   `svn co http://svn.apache.org/repos/asf/xmlbeans/tags/2.3.0`.
   Some firewalls may block access to the repository.
   In the example below, this code is placed at c:\dev\xmlbeans\2.3.0.

3. Configure environment variables. For example:
   - `SET path=%path%;c:\dev\apache-ant-1.7.1\bin;`
   - `SET JAVA_HOME=c:\dev\Java\jdk1.5.0_07`
   - `SET XMLBEANS_HOME=c:\dev\xmlbeans\2.3.0`

4. Apply the `patch_alt_parser.txt` patch. If you are using the cygwin patch facility, use something like
   ```
   dos2unix patch_alt_parser.txt
   patch
   xmlbeans/2.3.0/src/store/org/apache/xmlbeans/impl/store/Locale.java < patch_alt_parser.txt
   ```

5. Build the file xbeans.jar from the build directory, using the commands
   ```
   cd 2.3.0
   ant deploy
   ```
   These commands build the file `xbeans.jar` in the `build\ar` directory. You can test that the build ran successfully by running:
   ```
   ant checkintest
   ```

To use the Intel® XML Parsing Accelerator in XMLBeans:

1. Replace the old `xbean.jar` file with a new file. You may find it in places such as:
   `\bea\weblogic92\common\lib\apache_xbean.jar` for Oracle WebLogic Server* 9.2

2. Configure environment variables, and set the system properties to use the Intel® XML Parsing Accelerator. For example, these can be put in ..\user_projects\domains\DomainIntermediary\bin\setDomainEnv.cmd for Oracle AquaLogic Service Bus 2.5
   - `SET JAVA_OPTIONS=%JAVA_OPTIONS%`
     `-Djava.library.path=`
     `<install_path>\XMLSoftwareSuite\Java\1.2\Bin\ia32`
     `-Dorg.apache.xmlbeans.xmlreader=`
     `com.intel.xml.sax.XMLReaderImpl`

   - SET `CLASSPATH=%CLASSPATH%;<install_path>\XMLSoftwareSuite\Java\1.2\Lib\intel-xss.jar;`
     To make the Intel XML Software Suite the default processor for only parsing and not all XML processing, append the Intel XML Software Suite to the end of the Java `CLASSPATH`.

3. Adjust the configurations above for your applications that use XMLBeans. For example, Oracle AquaLogic Service Bus 2.5 has an additional library path, so replace the corresponding line above with: `-Djava.library.path=<install_path>\XMLSoftwareSuite\Java\1.2\Bin\ia32;<install_path>\bea\weblogic92\server\native\win\32`

3 Using the Intel® XML Parsing Accelerator

The Intel® XML Parsing Accelerator for Java* Environments performs parsing of input XML data following the SAX or DOM model, and can evaluate input data against a set of constraints by using the Intel® XML Schema Accelerator.

Parsing Data in SAX Mode

When parsing the document in SAX mode, you need to define the set of actions to be executed when a certain part of an XML document is parsed. The Intel® XML Parsing Accelerator supports:

- the SAX2 `rg.xml.sax.ContentHandler` interface
- the `org.xml.sax.DocumentHandler` interface compatible with SAX 1.0 standard [5].

The main difference between the two interfaces is that `ContentHandler` provides namespace support while `DocumentHandler` does not.

To parse a document in SAX mode:

1. Implement an event handler.
   To implement and register an event handler, create a new class inheriting from the class `org.xml.sax.helpers.DefaultHandler`. The `DefaultHandler` class provides all the required methods for parsing events. Your code can inherit from this class and implement the methods your application actually uses. See Creating a New Handler for Data Processing.
2. Get a `SAXParserFactory` instance.
   The Intel XML Parsing Accelerator implements a factory compatible with `javax.xml.parsers.SAXParserFactory` for all `SAXParser` objects. To obtain a new `SAXParserFactory`, call the `newInstance()`method.

3. Create the parser.
   The `newSAXParser()` method in the `SAXParserFactory` class provides a new SAXParser object that you can use.

4. Perform the SAX parsing process with the class you created in step 1.
   The `parse()` method of the `SAXParser` instance parses an XML source; as a result, the event methods in the inherited `DefaultHandler` class are called.

5. [optional] Reset the parser.
   The `reset()` method in the `SAXParser` class can reset the parser for further XML processing.

The sample code below illustrates these steps:

Creating a New Handler for Data Processing

```
import org.xml.sax.helpers.DefaultHandler
class UserDefinedSAXEventHandler extends DefaultHandler {
   public void startElement(String uri,
   ...
}
```

Parsing an XML File in SAX Mode

```
// Get SAX Implementation Factory, which is managed
// by the system; do not release it.
SAXParserFactory saximpl = SAXParserFactory.newInstance();
//Create a SAXParser object from SAXFactory.
SAXParser parser = saximpl.newSAXParser();
//Set user-defined EventHandler into SAXparser.
UserDefinedSAXEventHandler userDefinedSAXEventHandler = new
UserDefinedSAXEventHandler();
// Parse your XML file through the file
parser.parse(xmlFileName, userDefinedSAXEventHandler);
// (optional) Reset the SAXParser instance created by
SAXParserFactory.
parser.reset();
```

Parsing Data in DOM Mode

To parse your input data with the Intel® XML Parsing Accelerator:

1. Get a `javax.xml.parsers.DocumentBuilderFactory` object by calling the `newInstance()`static method.

2. Create the parser.

   The `newDocumentBuilder()` method in the `javax.xml.parsers.DocumentBuilderFactory` class provides the new `DocumentBuilder` object that you can use.

3. Perform the DOM parsing process.

   The `parse()` method of the `DocumentBuilder` object works over XML source and creates a tree view `Document` object. The source data processed by the parser can be of the following types:

   - An XML file
   - URI of an XML file
   - `InputSource`
   - `InputStream` from XML content or a file

The Parsing an XML file to a DOM Tree example illustrates these steps with sample code.

Parsing an XML file to a DOM Tree

```
import javax.xml.parsers.DocumentBuilderFactory; import
javax.xml.parsers.DocumentBuilder; import org.w3c.dom.Document;

//Get a DocumentBuilderFactory instance. DocumentBuilderFactory
factory = DocumentBuilderFactory.newInstance();

//Create a DocumentBuilder form DocumentBuilderFactory.
DocumentBuilder builder = factory.newDocumentBuilder();

// Parse your XML file to a DOM tree through the file URI.
Document doc = builder.parse(new File(xmlFileName));
```

NOTE

`Document.destroy()`method provides a performance benefit for all Java* DOM parsing XML files. For intensive workloads, such as repeated Java* DOM parsing of large XML files during a prolonged amount of time by many threads, performance of the product may degrade due to extensive memory usage. To avoid this, explicitly invoke `Document.destroy()` to do cleanup and set the Document Object to NULL, as shown in the Tuning for Intensive Workloads example. In this case, the application code notifies the JVM* that the Document object returned by `DocumentBuilder.parse()` will not be used any more.

`Document.destroy()` can provide a more significant performance boost for XML files larger than 640K, and more than 2 threads or iterations.

Tuning for Intensive Workloads

```
try {
// find the method "destroy()" and invoke it to do the clean up

   Class docClass = doc.getClass();
   Method dtor = docClass.getMethod("destroy", new Class[0]);
   if (dtor != null) {
     dtor.invoke(doc, new Object[0]);
     }
}
catch (Exception e) {}
// notify JVM that this object will not be used any more
doc = null;
```

Accessing and Updating the DOM Tree

To access and update the DOM tree created by a `DocumentBuilder`, use the DOM API methods. For a detailed description of these methods, see the API documentation, `Index_API_Java.html`.

NOTE

To achieve a better performance for document parsing, set the `http://apache.org/xml/features/dom/defer-node-expansion` feature to TRUE in `DocumentBuilderFactory`. In this mode, the read-only DOM APIs are not thread-safe because of the lazy expansion of DOM nodes.

To achieve thread-safe DOM tree traversal, set this feature to FALSE.

The Editing a Document Object example demonstrates creating a new element with an `employeeId` local name belonging to the `test` namespace, and then appending the element to a document object.

Editing a Document Object

```
import org.w3c.dom.Element; import org.w3c.dom.Node;

// Access to the child node that is the root element of the
document doc. Element rootElement = doc.getDocumentElement();

// Create an element object. Element element =
doc.createElementNS("test","employeeId");

// Append the element object as the last child of the root
element. Node appendedChild = rootElement.appendChild(element);
```

Parsing and Writing Data in StAX Mode

The Streaming API for XML includes two API sets: the *cursor API* and the *iterator API*.

The cursor API enables efficient reading and writing of the XML data. It has the `XMLStreamReader` and `XMLStreamWriter` interfaces.

The iterator API enables easy pipelining. It is an easy–to-use, easy-to-extend event-based API with `XMLEventReader` and `XMLEventWriter` interfaces.

To parse a document in StAX mode:

1. Call the `newInstance()` method to get an `XMLInputFactory` instance. The Intel XML Parsing Accelerator implements a factory compatible with `javax.xml.stream.XMLInputFactory`.
2. Call the `createStreamReader()` or `createEventReader()` methods for XML input to create `XMLStreamReader` or `XMLEventReader`.
3. Iterate the `XMLStreamReader` or `XMLEventReader` objects to get XML document information.

Example 1 and Example 2 illustrate document parsing with the sample code.

To write a document in StAX mode:

1. Call the `newInstance()` method to get an `XMLOutputFactory` instance. The Intel XML Parsing Accelerator implements a factory compatible with `javax.xml.stream.XMLOutputFactory`.
2. Call the `XMLOutputFactory createStreamWriter()` or `createEventWriter()` methods with an output stream to create `XMLStreamWriter` or `XMLEventWriter`.
3. Call the methods of the `XMLStreamWriter` or `XMLEventWriter` objects to write the XML document to the output stream.

Example 3 and Example 4 illustrate document parsing with the sample code.

Example 1. Parsing an XML File with StAX Cursor API

```
// Get XMLInputFactory, which is managed by the system; do not
// release it.
XMLInputFactory InputFactory = XMLInputFactory.newInstance();
//Create an XMLStreamReader object from InputFactory.
XMLStreamReader reader =
InputFactory.createStreamReader(xmlFileName);
```

```
//Iterate on the document with XMLStreamReader and get the
information of the document according to current status
while (reader.hasNext()) {
        reader.next();
        int nextEvent = reader.getEventType();
                //Your processing code
        switch (nextEvent) {
        case XMLStreamReader.START_ELEMENT:
             String nameStartElement =reader.getLocalName();
             ……
             break;
         case XMLStreamReader.END_ELEMENT:
             String nameEndElement = reader.getLocalName();
             ……
             break;
         case XMLStreamReader.CHARACTERS:
             String text = reader.getText();
             ……
             break;
         case XMLStreamReader.PROCESSING_INSTRUCTION:
             String target = reader.getPITarget();
             ……
             break;
       ……
       ……
        default:
             break;
      }
}
// Frees any resources associated with this Reader
reader.close();
```

Example 2. Parsing an XML File with StAX Iterator API

```
// Get XMLInputFactory, which is managed by the system; do not
// release it.
XMLInputFactory InputFactory = XMLInputFactory.newInstance();
//Create an XMLEventReader object from InputFactory.
```

```
XMLEventReader reader =
InputFactory.createEventReader(xmlFileName);
//Iterate on the document with XMLEventReader and get the
information of the document from the events
while (reader.hasNext()) {
       XMLEvent event = reader.nextEvent();
       switch (event.getEventType()) {
       case XMLEvent.START_ELEMENT:
               QName nameStartElement =
event.asStartElement().getName();
               ……
               break;
       case XMLEvent.END_ELEMENT:
               QName nameEndElement =
event.asEndElement().getName();
               ……
               break;
       case XMLEvent.CHARACTERS:
               String text = event.asCharacters().getData();
               ……
               break;
       case XMLEvent.PROCESSING_INSTRUCTION:
               String target =
((ProcessingInstruction)event).getTarget();
               ……
               break;
       default:
               break;
       }
}
//Frees any resources associated with this Reader
reader.close();
```

Example 3. Writing an XML File with XMLStreamWriter

```
         OutputStream out = null;
        try {
            out = new FileOutputStream("data.xml");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
```

```
        }
        try {
            // XMLStreamWriter is created by XMLOutputFactory
            XMLOutputFactory factory =
XMLOutputFactory.newInstance();
            XMLStreamWriter writer = null;
            writer = factory.createXMLStreamWriter(out, "ISO-
8859-1");
            // write XML file with writeFoo... methods
            writer.writeStartDocument("ISO-8859-1", "1.0");
            writer.writeComment("This is a simple StAX
streamWriter sample");
            writer.writeStartElement("greeting");
            writer.writeAttribute("id", "g1");
            writer.writeCharacters("Hello StAX");
            writer.writeEndDocument();
            // flush and close the writer
            writer.flush();
            writer.close();
            out.close();
        } catch (XMLStreamException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Example 4. Writing an XML File with XMLEventWriter

```
    /** Load the input file */
    public byte[] loadFile(String filename){
        byte[] buffer = new byte[1024];
        FileInputStream infile = null;
        try {
            infile = new FileInputStream(filename);
        }
        catch (FileNotFoundException e) {
            System.err.println("Can not find the file:" +
filename);
            System.exit(1);
        }
```

```
        try {
            int count = 0;
            ByteArrayOutputStream baos = new
ByteArrayOutputStream();
            while((count = infile.read(buffer)) != -1) {
                baos.write(buffer, 0, count);
            }
            infile.close();
            return baos.toByteArray();
        } catch (Exception e) {
            System.err.println("File error!");
            System.exit(1);
        }
        return null;
    } //loadFile(String)


    /** Use eventReader to process the input file and collect the
event */
    private void genEvent(byte[] xml) {
        list.clear();
        InputStream stream = new ByteArrayInputStream(xml);
        XMLEventReader reader;
        try {
            XMLInputFactory InputFactory =
XMLInputFactory.newInstance();
            reader = InputFactory.createXMLEventReader(stream);
            while (reader.hasNext()) {
                list.add(reader.nextEvent());
            }
        } catch (XMLStreamException e) {
            e.printStackTrace();
        }
    } //genEvent(byte[])
    /** The write process */
    public void write() {
        // use eventReader to process the input file and get
events.
        genEvent(loadFile(fileName));
        Boolean startDocWritten = false;
```

```
        // Create XMLEventWriter by XMLOutputFactory
        XMLOutputFactory OutputFactory =
XMLOutputFactory.newInstance();
        OutputStream stream = new ByteArrayOutputStream();
        XMLEventWriter writer = null;
        try {
            writer = OutputFactory.createXMLEventWriter(stream,
"UTF-8");
        } catch (XMLStreamException e) {
            e.printStackTrace();
        }
        // use .add() method to write
        try {
            for (int i = 0; i < list.size(); i++) {
                XMLEvent event = list.get(i);
                int type = event.getEventType();
                if (type == XMLStreamReader.START_DOCUMENT) {
                    if (!startDocWritten) {
                        writer.add(event);
                        startDocWritten = true;
                    }
                }
                else if (type != XMLStreamReader.END_DOCUMENT)
                    writer.add(event);
            }
            stream.flush();
            writer.add(XMLEventFactory.newInstance().createEndDoc
ument());
            stream.flush();
        } catch (XMLStreamException e) {
            e.printStackTrace();
        } catch (IOException e1) {
            e1.printStackTrace();
        }
        // Output the write result.
        System.out.println(stream.toString());
    } //write()
```

Enabling Data Validation

The Intel® XML Parsing Accelerator can operate in a validating or a non-validating mode. To use the Intel XML Schema Accelerator to validate input data, do one of the following before parsing:

- For the SAX mode, call the `setSchema()` method in the instance of the `SAXParserFactory` class. For the DOM mode, call the in the instance of the `DocumentBuilderFactory` class.
- For the SAX mode, set the `schemaLanguage` or `schemaSource` properties in the instance of the `SAXParserFactory` class. For the DOM mode, set the same properties in the instance of the `DocumentBuilderFactory` class.

The syntax for enabling input data validation differs for SAX and DOM parser operating modes. The DOM Mode: Enabling Validation by Calling setSchema Method and DOM Mode: Enabling Validation by Setting Properties examples show how to enable validation when parsing in the DOM mode. The SAX Mode: Enabling Validation by Calling setSchema Method and SAX mode: Enabling Validation by Setting Properties examples show how to enable validation when parsing in the SAX mode.

DOM Mode: Enabling Validation by Calling setSchema() Method

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
SchemaFactory sfactory =
SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
Schema schema = sfactory.newSchema(new File(schemaFile));
factory.setSchema(schema);
DocumentBuilder parser = factory.newDocumentBuilder();
Document document = parser.parse(new File(xmlFile));
```

SAX Mode: Enabling Validation by Calling setSchema() Method

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SchemaFactory sfactory =
SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
Schema schema = sfactory.newSchema(new File(schemaFile));
factory.setSchema(schema);
SAXParser parser = factory.newSAXParser();
parser.parse(new File(xmlFile),userDefinedSAXEventHandler);
```

DOM Mode: Enabling Validation by Setting Properties

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
factory.setValidating(true);
factory.setNamespaceAware(true);
factory.setAttribute("http://java.sun.com/xml/jaxp/properties/sch
emaSource"
"file:///C:/temp/technical documents/JAXP examples/1.xsd");
DocumentBuilder parser = factory.newDocumentBuilder();
Document document = parser.parse(new File("C:/temp/technical
documents/JAXP examples/1.xml"))
```

SAX mode: Enabling Validation by Setting Properties

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true);
factory.setNamespaceAware(true);
.SAXParser parser = factory.newSAXParser();
parser.setProperty("http://java.sun.com/xml/jaxp/properties/schem
aLanguage",
"http://www.w3.org/2001/XMLSchema");
parser.setProperty("http://java.sun.com/xml/jaxp/properties/schem
aSource",
"file:///C:/temp/technical documents/JAXP examples/1.xsd");
parser.parse (new File("C:/temp/technical documents/JAXP
examples/1.xml"),userDefinedSAXEventHandler
```

For details on the Intel XML Schema Accelerator specifics, see Using the Intel® XML Schema Accelerator.

Enabling DTD Validation

The Intel® XML Parsing Accelerator can validate the DTD during SAX mode and DOM mode parsing.

Enabling DTD Validation in DOM Mode

In DOM mode, the Intel XML Parsing Accelerator uses the default Apache* Xerces-J* SAXParser to perform DTD validation and the Intel DOM implementation to parse the data. DTD validation is performed when the following conditions are met:

- The XML input file provides the system ID of the DTD file.
- The method `DocumentBuilderFactory.isValidate()` returns TRUE.

Example

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
factory.setValidating(true);
DocumentBuilder parser = factory.newDocumentBuilder();
Document document = parser.parse(new File(xmlFile));
```

Enabling DTD Validation in SAX Mode

In SAX mode, the Intel XML Parsing Accelerator uses the default Apache* Xerces-J* SAXParser to perform DTD validation and data parsing. DTD validation is performed when the following conditions are met:

- The method `setValidating()` returns TRUE.

- The property `http://java.sun.com/xml/jaxp/properties/schemaLanguage` is not set to `http://www.w3.org/2001/XMLSchema`.
- The property `http://java.sun.com/xml/jaxp/properties/schemaSource` is not set.

Example

```
SAXParserFactory saximpl = SAXParserFactory.newInstance();
saximpl.setValidating(true);
SAXParser parser = saximpl.newSAXParser();
UserDefinedSAXEventHandler userDefinedSAXEventHandler = new
UserDefinedSAXEventHandler();
parser.parse(xmlFileName, userDefinedSAXEventHandler);
```

4 Using the Intel® XML Schema Accelerator

The Intel® XML Schema Accelerator for Java* Environments verifies input XML data against a *schema.* A schema is a set of constraints that can be checked or enforced against an XML document. Invoke the Intel XML Schema Accelerator through the Intel® XML Parsing Accelerator when performing DOM and SAX parsing, as described in section Using the Intel® XML Parsing Accelerator.

Validating Data

To run the Intel® XML Schema Accelerator for Java* Environments against your XML data:

1. Get a `SchemaFactory` object.

   To abstract your application code from the schema implementation, the JAXP interface provides an abstract `SchemaFactory` class. The `newInstance()`static method should be called on this class to obtain a factory. A `SchemaFactory` object is:

   - Not thread-safe: application code must ensure that only one thread uses a `SchemaFactory` object at any given moment. Implementations are encouraged to mark methods as synchronized to prevent exceptions being generated.
   - Not re-entrant: application code cannot recursively invoke the `newSchema()` method while it is already invoked, even in the same thread.

2. Load and compile the schema to be used.

   The schema can be loaded from a file in a `java.io.File` object, a remote URL in a `java.net.URL` object, or an XML transformation in a `javax.xml.transform.Source` object.

   If the schema location is not specified, the JAXP interface provides the following method for creating a new schema:

```
SchemaFactory factory =
SchemaFactory.newInstance(http://www.w3.org/2001/XMLSchema)
;
```

```
Schema schema = factory.newSchema();
...
```

NOTE

You could also specify the location of the schema using the attributes `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation` inside the XML data. However, this method is not recommended because such schema location hints can introduce a vulnerability to denial-of-service attacks.

3. Get a validator from the schema.

   Parse the document you need to check by creating a `javax.xml.transform.Source` object for the instance document.

4. Validate your data.

If the source data is valid, the library displays nothing. If the data is invalid, the Intel XML Schema Accelerator throws a `SAXException` object or reports errors to the specified custom `ErrorHandler`. The Validation Procedure example illustrates how to check data with the Intel XML Schema Accelerator.

The current version of the Intel XML Schema Accelerator does not support PSVI, so the output is not accompanied by augmented information.

Validation Procedure

```
import java.io.*;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.*;
import org.xml.sax.SAXException;


public class DocbookXSDCheck {


    public static void main(String[] args) throws SAXException,
            IOException {


        // 1. Lookup a factory for the W3C XML Schema language
        SchemaFactory factory = SchemaFactory.newInstance
        ("http://www.w3.org/2001/XMLSchema");
        factory.setErrorHandler(new MyErrorHandler());


        // 2. Compile the schema.
        // Here the schema is loaded from a java.io.File,
        // but you could use
```

```
        // a java.net.URL or a javax.xml.transform.Source
        // instead.
        File schemaLocation =
        new File("/opt/xml/docbook/xsd/docbook.xsd");
        Schema schema = factory.newSchema(schemaLocation);


        // 3. Get a validator from the schema.
        Validator validator = schema.newValidator();


        // 4. Parse the document you want to check.
        Source source = new StreamSource(args[0]);


        // 5. Check the document
        try {
            validator.validate(source);
            System.out.println(args[0] + " is valid.");
        }
        catch (SAXException ex) {
            System.out.println(args[0] + " is not valid because
");
            System.out.println(ex.getMessage());
        }
    }
}
```

Configuring Validation

User applications can employ several means of adjusting the validation process, as follows:

Error Handling

When you set an error handler, errors found during parsing a schema or validating an XML document are first sent to the `ErrorHandler` object. The error handler can abort the parsing of a schema or validation of an XML document immediately by throwing `SAXException` from the handler.

You can specify your own error handler instead of the default one. For that, you need to define the following functions:

- `void error(SAXParseException)`

- `void fatalError(SAXParseException)`
- `void warning(SAXParseException)`

The Setting a Custom Error Handler example shows how you can set your own error handler.

Setting a Custom Error Handler

```
public class ForgivingErrorHandler implements ErrorHandler
  {  public void warning(SAXParseException
ex)     {     System.err.println(ex.getMessage());
    }  public
void error(SAXParseException ex)  {
    System.err.println(ex.getMessage());
    }  public
void fatalError(SAXParseException ex) throws SAXException     {

    throw ex;     }
 }
```

Resolving External Resources

To redirect an included or imported file or the specified schema location, you can define your own external resource resolver implementing the `org.w3c.dom.ls.LSResourceResolver` interface. For that, call method `resolveResource()`, which allows user code to resolve external resources and returns them as `org.w3c.dom.ls.LSInput`. Returned values can be `characterStream`, `byteStream`, `stringData`, `systemId`, and `publicId`.

5 Using the Intel® XSLT Accelerator

This section explains:

- how to transform XML data with the Intel® XSLT Accelerator
- how to customize the processing.

Performing the XSL Transformation

To transform your XML input data with the Intel XSLT Accelerator, do the following:

1. Instantiate a `TransformerFactory` object.

   To abstract your application code from the transformer implementation, the JAXP interface provides an abstract `TransformerFactory` class. To obtain a factory, call the `newInstance()` static method on this class.

2. Provide your XML data into the transformer.

   Calling the `newTransformer(Source xslSource)` method on the `TransformerFactory` class creates a `Tranformer` object. This method reads a supplied `Source` stylesheet and produces a `Transformer` object that you can use to perform the transformation.

   You can supply the stylesheet just as your input data: in a stream of XML markup (`StreamSource`), in a DOM node (`DOMSource`), SAX input (`SAXSource`) or StAX input (StAXSource).

3. Perform the transformation.

   The `transform(Source xmlSource, Result transformResult)` method of the `Transformer` object reads from the XML source and places the output of the transform in a `Result` object.

The XML source can be provided in the form of a `StreamSource`, `DOMSource`, or `SAXSource` object, and the output can be a `StreamResult`, `DOMResult`, `SAXResult`, or StAXResultobject.

The steps required to perform an XSL transformation are shown in the Using a Transformer to get a DOM Tree example and further illustrated in the `TransformExample` sample application supplied with the product. Consult Sample Applications for details on locating and running the example.

Using a Transformer to get a DOM Tree

```
// Generate a Transformer object.

tFactory = TransformerFactory.newInstance();
javax.xml.transform.Transformer transformer =
tFactory.newTransformer(new
javax.xml.transform.stream.StreamSource("foo.xsl"));

// Create an empty DOMResult object for the output.
javax.xml.transform.dom.DOMResult domResult = new
javax.xml.transform.dom.DOMResult();

// Perform the transformation. transformer.transform(new
javax.xml.transform.dom.DOMSource(inDoc), domResult);

// Get the output Node from the DOMResult.

org.w3c.dom.Node node = domResult.getNode();
```

You can also use an extension function to redirect your output into one or more files, see Other Extensions.

Customizing the Intel® XSLT Accelerator

Use the following options to customize the operation of the Intel® XSLT Accelerator:

- Set output properties in the `xsl:output` element of a stylesheet.

  After the stylesheet is compiled, the output properties specified in the XSLT stylesheet can be queried using the `getOutputProperties()` method on the `javax.xml.transform.Templates` or the `javax.xml.transform.Transformer` object. You can override output property values when performing a transformation by calling `setOutputProperty()` and `setOutputProperties()` on the `Transformer` object.

- Add parameters to your stylesheet during transformation.

  For that, the `setParameter()`method of the `Transformer` object is used. After setting a parameter, you can retrieve it using the `getParameter()` method.

- Set the mode property to specify the target behavior that Intel XSLT Accelerator conforms to. You can choose between 5 modes: Standard, MSXML, XALANJ, XSLTC, and XT. The default value is Standard. You can set this property as follows:

```
<component name ="Intel XSLT Accelerator">
      <property name="mode" value="XALANJ" />
</component>
```

In this example, the Intel XSLT Accelerator conforms to the Xalan-J processor behavior.

Object Types

Using Object Types

As shown in the transformation description, XML processing can involve data streams, DOM trees, SAX events, and StAX events. Each object type has its specifics in the transformation, as described below.

Streams

To create a `StreamSource` object, use a system ID, which is a filename following the URI syntax, or a `java.io.InputStream` or `java.io.Reader` object.

DOM Trees

Your transformations can involve `DOMSource` and `DOMResult` objects provided by the `javax.xml.transform.DOM` package. These involve operations with a DOM (document object model) tree. To transform `DOMSource` into a stream, create a new `Transformer` object and make a "copy" of your DOM tree as a stream of data.

To produce a `DOMResult` object out of a stream of data, use the `DocumentBuilderFactory` object, which creates a `DocumentBuilder` object for your needs. To transform your data into a DOM tree, create a new `DOMResult` object or use `DOMResult.setNode()` to assign a new container.

SAX Events

You can use SAX (Simple API for XML) events in your input data, source stylesheet instructions or output. In the transformation engine, the `SAXParser` interface defines several `parse()` methods to handle SAX events. When a `parse()` method is called, the parser invokes one of the callback handler methods in your application.

You can implement content handlers, error handlers, and other methods depending on your needs. Because `SAXParser` is a wrapper for the `SAXReader` object, you can easily plug in your own reader instead of it. For SAX-specific methods, you can use a `SAXTransformerFactory` object, see Java* API for XML Processing (JAXP) description [9]. This feature enables the use of XML filter to pass output of one transformation as input for another transformation via the `SAXTransformerFactory` `newXMLFilter(Source)` and `newXMLFilter(Templates)` methods.

StAX Events

You can use StAX (Streaming API for XML) events in your input data, source stylesheet instructions or output. You can use the XMLInputFactory object to create a

XMLEventReader or XMLStreamReader, which can then be used to construct a StAXSource. Similarly, StAXResult can be constructed with XMLOutputFactory. For StAX-specific methods, you can see the javax.xml.stream part of Java* API for XML Processing (JAXP) description [9].

XSLT Extensions

You can expand the functionality of XSLT transformations with XSLT extensions. An extension can be expressed as a function or an element. Intel® XSLT Accelerator provides a subset of the function and element extensions that are defined by the EXSLT community project [13]. These extensions are referred to as *EXSLT extensions*.

You can also define you own Java extension functions that can be invoked during a transformation. These extensions may be static or instance methods.

EXSLT Extensions

The current version of the Intel® XSLT Accelerator supports EXSLT extensions grouped into the following modules:

- Common functions
- Date-and-time functions
- Math functions
- Sets functions
- Strings functions

Common functions

These functions cover the basic operations. For example, the `common:object-type` function returns the type of the supplied object (see the Using Object Types section for a definition of processed types).

The current version of the Intel XSLT Accelerator supports all functions of this module as defined in the EXSLT resource [13]. In addition to that, the library supports the `nodeset` Xalan-J extension, which matches the `common:node-set` EXSLT extension function.

Date-and-time functions

These functions handle operations related to date and time; for example, the `date:hour-in-day` function returns the hour of the day as a number.

The current version of the Intel XSLT Accelerator supports all core functions and most other functions as defined in the EXSLT resource [13], specifically:

```
date:add
date:add-duration
date:date
date:day-abbreviation
date:day-of-week-in-month
date:day-in-month
date:day-in-week
date:day-in-year
date:day-name
date:difference
date:duration
date:format-date
date:hour-in-day
date:leap-year
date:minute-in-hour
date:month-abbreviation
date:month-in-year
date:month-name
date:second-in-minute
date:seconds
date:time
date:week-in-year
date:year
```

The current version of the library does not implement the following EXSLT functions of this module: `date:parse-date`, `date:week-in-month`, `date:sum`.

Math functions

These functions provide facilities for performing mathematical operations; for example, the `math:max` function returns the maximum value of the nodes passed as the argument.

The current version of the Intel XSLT Accelerator supports all functions of this module as defined in the EXSLT resource [13], specifically:

```
math:abs
math:acos
math:asin
math:atan
math:atan2
```

```
math:constant
math:cos
math:exp
math:highest
math:log
math:lowest
math:max
math:min
math:power
math:random
math:sin
math:sqrt
math:tan
```

Sets functions

These functions enable you to manipulate node sets. For example, the `set:difference` function gets nodes of two sets as arguments, compares them and returns the difference between the two sets.

The current version of the Intel XSLT Accelerator supports all functions of this module as defined in the EXSLT resource [13], specifically:

```
set:difference
set:distinct
set:has-same-node
set:intersection
set:leading
set:trailing
```

Strings functions

These functions are responsible for string manipulation. For example, the `str:tokenize` function splits up a string and returns a node set of token elements, each containing one token from the string.

The current version of the Intel XSLT Accelerator supports most functions of this module as defined in the EXSLT resource [13]; specifically:

```
str:align
str:concat
str:padding
str:split
str:tokenize
```

The current version of the library does not implement the following EXSLT functions of this module: `str:replace`, `str:encode-uri`, and `str:decode-uri`.

The current version of the Intel XSLT Accelerator does not support the following EXSLT modules: *dynamic*, *functions*, *random*, and *regular expressions*.

NOTE

The Intel XSLT Accelerator does not support alternative language implementations of EXSLT functions. For example, `{http://exslt.org/functions}:func` is not supported, so you cannot make use of implementations in JavaScript* technology.

The current version of the Intel XSLT Accelerator is fully compatible with the Apache Xalan XSLTC processor [11] in extension function support.

Using an EXSLT Function

For detailed instructions on how to use EXSLT functions, see the EXSLT website [13]. This section describes how to call an EXSLT function:

1. Declare a namespace referring to the EXSLT module that contains the desired function. For example, to call a `date-and-time` function, type:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:date="http://exslt.org/dates-and-times">
 ...
</xsl:stylesheet>
```

2. Call the extension function in your templates. For example, get the year as a number type from the input string, call the `date:year` function in the following way:

```
<xsl:value-of select="date:year()">
```

NOTE

By default, the extension namespace is output into the result tree. To prevent this, specify the `extension-element-prefixes` attribute value with the module namespace prefix.

User-Defined Extension Functions

User-defined extension functions enable you to augment transformations with custom functions. Extension functions can operate with several XSLT and non-XSLT object types as input arguments and return values. This section describes each object type in turn and lists the Java* argument types that the object type can map to.

Using Object Types

The Intel® XSLT Accelerator supports passing the following types of XSLT objects as arguments to extension functions:

- Simple types: string, Boolean or number
- Node-set types
- Result tree fragments as unchangeable sets of nodes

Table 5. Mapping XSLT Object Types to Java* Argument Types

XSLT Type	Java* Argument Type
Node-set	`org.w3c.dom.traversal.NodeIterator`
	`org.w3c.dom.Node or its subclasses`
	`org.w3c.dom.NodeList`
	`java.lang.String`
	`java.lang.Object`
	`char`
	`[double,float,long,int,short,byte]`
	`boolean`
Result tree fragment	Same as for the Node-set type.
String	`java.lang.String`
	`java.lang.Object`
	`boolean`
	`char`
	`[double,float,long,int,short,byte]`
Boolean	`boolean`
	`java.lang.String`

XSLT Type	Java* Argument Type
	`java.lang.Object`
	`java.lang.Boolean`
Number	`char`
	`boolean`
	`[double,float,long,int,short,byte]`
	`java.lang.String`
	`java.lang.Object`
	`java.lang.Double`

NOTE

Nodes passed to extension functions as node-sets or result tree fragments are read-only and cannot be modified by extension functions.

Extension functions can return the following value types:

- Simple XSLT types: strings, number types and Boolean values
- XSLT node-sets that correspond to `org.w3c.dom.traversal.NodeIterator` and `org.w3c.dom.Node` Java* return types
- `Result` tree fragments to be contributed to the XSLT result document that corresponds to `org.w3c.dom.DocumentFragment` Java type.

Non-XSLT types are Java* objects that are passed as stylesheet parameters or returned by extension functions. Non-XSLT object types are mapped to Java* arguments as follows:

- native types or superclasses
- `double`
- `float`
- `long`
- `int`
- `short`
- `char`

- byte
- java.lang.String

Using your Extension Function

To invoke a user-defined extension function:

1. Declare the namespace for your function(s), for example:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:my-class="xalan://java.lang.Integer">
```

   In addition to the abridged syntax, the Intel® XSLT Accelerator supports Apache* Xalan-Java* specific namespace declaration syntax for Java* extensions [10] defined in Table "Formats for Xalan Namespace Declaration".

2. Call the extension function in your templates, for example:

```
<xsl:variable name="new-pop"
select="my-class:valueOf("12345")"/>
```

NOTE

You can call an extension function inside another extension function both for user-defined and for EXSLT extension functions.

Table 6. Formats for Xalan* Namespace Declaration

Name	Format

Class format	`xmlns:my-class="xalan://full_class_name"`
	Where `full_class_name` is the fully qualified class name.
	Examples:
	`xmlns:my-class="xalan://java.util.Hashtable" xmlns:my-class="xalan://mypackage.myclass"`

Package format	`xmlns:my-package="xalan://package_name"`
	Where `package_name` is the beginning of the Java* package name.
	Examples:
	`xmlns:my-package="xalan://java.util" xmlns:my-package="xalan://mypackage"`
	Note that unlike Apache XSLTC, the Intel XSLT Accelerator has no concept of a

Name	Format
	default object for extension functions.
Java* format	`xmlns:java="http://xml.apache.org/xalan/java"`

Other Extensions

In addition to EXSLT and user-defined extensions, the Intel® XSLT Accelerator supports the redirect extension specific for the Apache* Xalan-Java* implementation [10], which enables you to redirect parts of the transformation output to one or more files. To see how this extension is used, run the `Redirect` application in the `Examples` directory. In the current version, this extension supports the following encodings:

- UTF-8
- US-ASCII
- ISO-8859-1
- UTF-16(UTF-16LE)
- UTF-16BE
- UTF-32(UTF-32LE)
- UTF-32BE

For instructions on running this example, see Sample Applications.

6 Using the Intel® XPath Accelerator

The Intel® XPath Accelerator for Java* Environments evaluates XPath expressions against input data. An *XPath* expression is similar to a path as in URL, and can include expressions to manipulate strings, numbers and Boolean values. For example, the expression `//book/title[author="Charles Dickens"]` finds the titles of books written by Charles Dickens in a given XML document.

Performing the XPath Evaluation

To evaluate your input data with the Intel XPath Accelerator:

1. Instantiate an `XPathFactory` object.

   To abstract your application code from the `XPath` implementation, use the `XPathFactory` abstract class from the JAXP interface. To obtain a factory, call the `newInstance()` or `newInstance(String url)` static method on this class. The optional parameter `uri` specifies the object model. The current implementation in the Intel XPath Accelerator only supports the W3C DOM object model, which is `URI DEFAULT_OBJECT_MODEL_URI`.

2. Create an `XPath` object.

   The `newXPath()` method on the `XPathFactory` class provides the new `XPath` object that you can use to evaluate an XPath expression.

   The `XPath` object can accept XPath expressions and perform evaluation, so it mainly supplies `compile()` and `evaluate()` methods, and provides an interface to set and get objects `XPathVariableResolver, XPathFunctionResolver, and NamespaceContext.`

3. [optional] Create an `XPathExpression` object.

   To compile an XPath expression, call `compile(java.lang.String expression)`. This method returns an `XPathExpression` object that can perform evaluation against an input document. This step is optional because you can evaluate input directly without creating an `XPathExpression` object.

4. Perform the evaluation.

   To evaluate an XPath expression against XML data, you can call `evaluate()` on a compiled `XPathExpression` object (see the previous step) or directly on an `XPath` object.

As input data, you can supply a `java.lang.Object` instance, which specifies the context node as a DOM node, or an `org.xml.sax.InputSource` instance, which specifies the input data as an XML stream. For DOM node context, the Intel XPath Accelerator supports DOM nodes created by the Intel® XML Parsing Accelerator and third-party DOM nodes.

The `evaluate()` method of the `XPath` and `XPathExpression` objects operates on the XML source and returns result objects `java.lang.String` or `java.lang.Object`. If the type of the returned output is not specified, the method produces the output as a `String` object. You can also specify the return type to get method output as an `Object` of one of the following types: `XPathConstants.BOOLEAN`, `XPathConstants.NUMBER`, `XPathConstants.STRING`, `XPathConstants.NODESET`, or `XPathConstants.NODE`.

NOTE

The `evaluate()` method of the `XPath` and `XPathExpression` objects is thread-safe. You can call it from multiple threads with different input data.

The Sample XPath Evaluation example demonstrates the steps of the evaluation process.

NOTE

The JAXP 1.3 specification [9] allows the context to be a node set. However, this does not conform to W3C XPath 1.0 recommendation [8]. In JAXP 1.4, the node set as context is removed. As it is difficult to provide a safe implementation of XPath processing of such input, the Intel® XPath Accelerator throws an exception if a node set is specified as the input context to avoid confusion.

Sample XPath Evaluation

```
// Generate an XPathFactory object.
XPathFactory factory = XPathFactory.newInstance();
// Generate an XPath object.
javax.xml.xpath.XPath xpath = factory.newXPath();
// Compile an expression.
javax.xml.xpath.XPathExpression xpathExp=xpath.compile("/a/b/c");
// Create a document builder.
DocumentBuilderFactory docFactory =
DocumentBuilderFactory.newInstance(); DocumentBuilder builder =
docFactory.newDocumentBuilder();
// Parse a document.
Document doc = builder.parse(new FileInputStream(new
File("test.xml")));
// Invoke the evaluation, the result will be stored in result
object. org.w3c.dom.NodeList result=xpathExp.evaluate(doc,
XPathConstants.NODESET);
```

Resolving External Resources

When the evaluated XPath expression references a variable or external function or contains a namespace prefix, the XPath evaluation process depends on resolvers set by the calling application. Before compiling and evaluating an XPath expression, the application uses relative methods to set the correct resolver and context. The variable resolver and function resolver can be set as properties of `XPathFactory` and `XPath` objects, and the namespace context can only be set as a property of `XPath`.

Namespace Context

When namespace prefixes are used in an XPath expression, the prefixes need to be resolved to URI addresses. The mapping of a namespace prefix to URI is provided by an object implementing the `javax.xml.namespace.NamespaceContext` interface.

You can set your derived `javax.xml.namespace.NamespaceContext` objects to `XPath` objects by calling the `XPath.setNamespaceContext()` method.

You need to implement one of the following methods:

- `java.lang.String getNamespaceURI(java.lang.String prefix)`
- `java.lang.String getPrefix(java.lang.String namespaceURI)`
- `java.util.Iterator getPrefixes(java.lang.String namespaceURI)`

The User-defined Namespace Context example demonstrates setting the namespace prefix mapping through the `getNamespaceURI()` method.

User-defined Namespace Context

```
public class MyNamespaceContext implements NamespaceContext
{
public String getNamespaceURI(String prefix)
  {
     if (prefix.equals("user"))
         return "http://user.com";
     else
         return null;
   }
   public String getPrefix(String namespace)
   {
     if (namespace.equals("http://user.com"))
         return "user";
     else
```

```
        return null;
   }
   public Iterator getPrefixes(String namespace)
   {
   return null;
   }
  }
```

Variable Resolver

When variables are used in an XPath expression, the Intel® XPath Accelerator needs to get the type and value of these variables. For that, JAXP enables you to implement an `XPathVariableResolver` object, and to set the object via `setXPathVariableResolver()`. `XPath` and `XPathFactory` objects have this method. If the `XPathVariableResolver` is set on `XPathFactory`, all `XPath` objects constructed from this `XPathFactory` use the specified `XPathVariableResolver` by default.

The Intel XPath Accelerator uses the `XPathVariableResolver` object to retrieve the value of a user-defined variable. A value of a variable must not change during the evaluation process. The `XPathVariableResolver` interface provides method `java.lang.Object resolveVariable(QName variableName)` returning the variable value according to the input name.

For example, if you want to evaluate an XPath expression `//book[name=$var]`, you need to implement an `XPathVariableResolver`, from which the `$var` can be retrieved, see the User-defined XPath Variable Resolver example.

User-defined XPath Variable Resolver

```
public class MyVarResolver implements XPathVariableResolver
{
public Object resolveVariable(QName qName)
{  if (qName.getLocalPart().equals("var"))
    return "dictionary";
else
    return null;
  }
}
```

Function Resolver

When user-defined functions are used in an XPath expression, the Intel® XPath Accelerator needs to access the value returned by the function. JAXP allows you to implement an `XPathFunctionResolver` object and set the object via `setXPathFunctionResolver()` for these purposes. Both `XPath` and `XPathFactory` objects have this method. If the `XPathFunctionResolver` is set on the `XPathFacory`, all `XPath` objects constructed from this `XPathFactory` use the specified `XPathFunctionResolver` by default. The Intel XPath Accelerator uses the `XPathFunctionResolver` to retrieve the value of a user-defined function.

The `XPathFunctionResolver` interface provides the `XPathFunction resolveFunction(QName functionName, int arity)` method returning the `XPathFunction` object according to the input function name and its arity. Object `XPathFunction` also requires you to provide the implementation, so that you could wrap the function code into an `XPathFunction object.`

For example, if you want to evaluate an XPath expression "`user:maximum(3,4)`", a `XPathFunctionResolver` sample could be the one shown in the User-defined XPath Function Resolver example.

User-defined XPath Function Resolver

```
public class MyFunctionResolver implements XPathFunctionResolver
 {
 public XPathFunction resolveFunction(QName fname, int arity)
 {
   if (fname.equals(new QName("http://user.com", "maximum",
"user")))
      return new XPathFunction() {
        public Object evaluate(java.util.List args) {
          if (args.size() == 2) {
              Double arg1 = (Double)args.get(0);
              Double arg2 = (Double)args.get(1);
              if(arg1>=arg2)
                return arg1;
              else
                return arg2;
             }
          else
             return null;
        }
     };
  else
    return null;
 }
}
```

7 Troubleshooting

This section describes the errors that you might get and their typical solutions.

Exception in thread "main" javax.xml.transform.TransformerFactoryConfigurationError: Provider com.intel.xml.transform.TransformerFactoryImpl not found

Verify that your CLASSPATH environment variable setting contains the path to jar files of the Intel® XML Software Suite for Java* Environments. For details on setting the required CLASSPATH, see Quick Start. You can confirm the values for the CLASSPATH environment variable on the command line, for example:

- On Linux* OS: `echo $CLASSPATH`
- On Windows* OS: `echo %CLASSPATH%`

Exception in thread "main" java.lang.UnsatisfiedLinkError: no intel-xss-j in java.library.path

Verify that your environment variables are configured to point to the correct locations. For details on setting the required variables, see Quick Start. You can confirm the values for the path variables on the command line, for example:

- On Linux* OS: `echo $LD_LIBRARY_PATH`
- On Windows* OS: `echo %PATH%`

Exception in thread "main" java.lang.UnsatisfiedLinkError: /home/java-api/jni/lib/libintel-xss-j.so: cannot open shared object file: No such file or directory java.lang.UnsupportedClassVersionError: unsupported classversion 49.0

Verify that you are using a JDK* or JRE* environment compatible with the Intel® XML Software Suite. Please consult the *Installation Guide* for a list of supported JDK and JRE versions. Check that your architecture is supported by this JDK or JRE environment.
The Intel XML Software Suite runs on 1.5.0 or above JDK and JRE environments. Check the version of Java* installation by running:
`java -version`

During compilation of an example from the Apache* Xalan Java* website, class `org.apache.xml.serializer.Serializer` is not found.

Make sure that the provided `serializer.jar` file is in the value of the `CLASSPATH` environment variable.

During execution of the example, the following error is produced: `java.lang.NoClassDefFoundError: org/apache/xml/serializer/...`

Make sure that the provided `serializer.jar` file is in the value of the `CLASSPATH` environment variable.

Need to check whether Intel XML Software Suite is used for processing.

Use the JVM option `-Djaxp.debug=1` or the Java option `-verbose:class`.
If your XML transformation is based on JAXP interfaces and the properties are set correctly, you should see the binding information printed on the console.

On Linux*, JVM crashed with an error message "Program terminated by signal number 11".

This may be caused by the incorrect setting of LD_PRELOAD. Verify that your LD_PRELOAD environment variable setting contains the path to libjsig.so of your JVM. For details on setting this environment variable, see *Quick Start*.

You can confirm the value of the LD_PRELOAD environment variable on the command line, for example:

```
echo $LD_PRELOAD
```

Appendix A: Acronyms and Definitions

This section lists the acronyms used throughout the document with their definitions.

Table 5. Acronyms Used in this Document

Acronym	Definition
API	Application programming interface
DOM	Document object model
EXSLT	Extensions to XSL Transformation
I/O	Input and output
JAXP	Java* API for XML processing
JDK*	Java* development kit
JNI	Java* native interface
JRE*	Java* run-time environment
PSVI	Post-schema-validation infoset
SAX	Simple API for XML
TrAX	Transformation API for XML Processing

Acronym	Definition
XML	Extensible markup language
XPath	XML Path Language
XSL	Extensible stylesheet language
XSLT	XSL transformation
StAX	Streaming API for XML

Appendix B: Compatibility Modes: Support for various Interpretations of XSL Conditions

This section provides details on the Intel® XSLT Accelerator compatibility modes: Standard, XT, MSXML, Xalan, and XSLTC.

These modes allow you to simulate different XSL behavioral interpretation of third party XSLT processors, such as Saxon, MSXML, Xalan, XT, and XSLTC. You can switch the compatibility mode on and off by setting property parameters in the `config.xml` configuration file, located in the `<install_dir>/conf` directory.

Table 6. Compatibility Modes: Support for various Interpretations of XSL Conditions

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
Local variable override	Two variables or parameters with the same name inside a template cause an error.	A local variable inside the scope of another local variable can have the same name. In this case, the inner variable shadows the outer variable; the outer variable retains its value.	Allowed in all modes: Standard, XT, Xalan, MSXML, and XSLTC.
Global variable override	A stylesheet with more than one binding of a global variable with the same name and import precedence causes an error.	You can declare a global variable more than once. The last declaration takes precedence.	Allowed in the Standard, XT, XalanJ and XSLTC modes. N/A in the MSXML mode.
Named template override	A stylesheet with more than one	More than one named template can have the	Allowed in all modes.

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
	template with the same name and import precedence causes an error.	same name. The last declaration takes precedence.	
Unknown variable	A variable not declared in the scope causes a static error.	You can use an undeclared variable if you skip static checking. An error only occurs when the variable's value is explicitly used at run time.	Allowed in the XT mode only. N/A in other modes.
Unknown named template	Unknown named template.	You can use an undeclared named template if you skip static checking. An error only occurs when the template is called during transformation.	Allowed in all modes.
Single right curly brace escape in AVT	Using a single right curly brace outside the expression in AVT causes an error. Use double left or right curly brace instead. They are automatically replaced by a single curly brace.	You can use a single right curly brace outside the expression. The result is the same as with two right curly braces.	Allowed in the XT mode only. N/A in other modes.
Complex content in comment PI and attribute construction	If xsl:processing-instruction and xsl:attribute create nodes other than text ones while	• If the offending node is a literal result element, the node and its content are ignored in the	• Allowed with implementation defined behavior in the MSXML mode.

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
	instantiating the content of xsl:comment, an error occurs. The XSLT processor may signal an error or recover by ignoring the offending nodes and their content.	MSXML mode. In other modes an error occurs. • If the offending node is constructed using xsl:element, the element is converted to a text node with the string-value of this element node. This is relevant for all modes. • If xsl:processing-instruction contains top-level XSLT instructions, an error occurs in the MSXML mode.	• Partially allowed with implementation defined behavior in Standard, XT, XalanJ and XSLTC modes. See the **Relax Rules** column for details.
Unrecognized top-level XSLT instruction	Only the XSLT elements defined in the specification and elements with non-null URI from other namespaces can be top-level elements.	• In the MSXML mode, an unrecognized top-level XSLT instruction causes an error. • In other modes, the unrecognized element is ignored.	Not allowed in the MSXML mode. Allowed in Standard, XT, XalanJ and XSLTC modes.

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
Unrecognized local XSLT instruction with forward compatibility	If an element in a template is processed in forwards-compatible mode (with version>1.0), and XSLT 1.0 does not allow to initiate such elements, XSLT performs fallback.	In the modes other than MSXML, an element not allowed in XSLT 1.0 causes an error if initiated at run time.	Not allowed in Standard, XT, XalanJ and XSLTC modes. N/A in the MSXML mode.
Invalid top-level element	Only the XSLT elements defined in the specification and elements with non-null URI from other namespaces can be top-level elements.	• In the MSXML mode, a top-level element outside the XSLT namespace with null URI.causes an error. • In other modes, the element is ignored.	Allowed in Standard, XT, XalanJ and XSLTC modes. Not allowed in the MSXML mode.
Unrecognized attribute on XSLT instruction	An XSLT instruction may have attributes not from other namespaces, if the expand-name of the attribute has a non-null namespace URI. Attributes with null namespace URI (for example, attributes with unprefixed	• If xsl:comment has any attribute in the MSXML mode (with both null and non-null URI), an error occurs. • If xsl:stylesheet or xsl:transform has an attribute 'xsl:version' instead of	Not allowed with implementation-defined behavior in the MSXML mode. N/A in other modes.

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
	names) not defined in the specification, cause an error for an XSLT instruction.	'version' in MSXML mode, an error occurs.	
Unrecognized XSLT attribute on literal result element	The literal result element has no attributes from the XSLT namespace.	The attributes in XSLT namespace not defined in the specification are ignored in the MSXML mode.	Allowed in all modes.
Invalid attribute value on XSLT instruction	The specification defines the value range or specifics of a certain attribute on an XSLT instruction.	• In the MSXML mode, an error occurs when: 1. xsl:output, method value is neither xml/html/text nor a valid QName 2. xsl:output 'standalone' is not defined 3. xsl:output 'indent' is not defined 4. xsl:template 'priority' is an empty string 5. xsl:sort 'data-type' is neither a number nor a text 6. xsl:sort 'order'	Allowed with implementation-defined behavior in Standard, XT, XalanJ, and XSLTC modes. Partially not allowed with implementation-defined behavior in the MSXML mode. See the **Relax Rules** column for details.

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
		is neither ascending nor descending 7. xsl:sort 'case-order' is neither upper-first nor lower-first • In other modes, the attribute is not specified, and the errors described above are ignored.	
xsl:element 'name' is not a QName	If the string resulting from instantiating 'name' attribute is not a QName, an error occurs. An XSLT processor may signal the error or recover by ignoring the element itself and instantiate its content.	In the MSXML mode, the XSLT processor signals an error. In other modes, it recovers.	Allowed in Standard, XT, XalanJ and XSLTC modes. Not allowed in the MSXML mode.
Non-white space texts in xsl:attribute-set	XSLT instructions have restrictions on their contents.	In the MSXML mode, an error occurs when xsl:attribute-set contains non-white space texts.	Allowed in Standard, XT, XalanJ and XSLTC modes. Not allowed in the MSXML mode.
Invalid pattern	If the pattern specified in the template is	• In the MSXML mode, the pattern error is	Allowed in Standard, XT, XalanJ, and XSLTC

XSL Condition	Specification Rules	Relax Rules	Mode Restrictions
	invalid, an error occurs.	checked statically. If the pattern is invalid, it processor signals an error. • In other modes, the behavior is undefined.	modes. Not allowed in the MSXML mode.
Output Namespace Alias stylesheet-prefix	Undefined in the specification.	The namespace specified as the stylesheet-prefix can be the output in all the modes, except for the MSXML mode.	Allowed in Standard, XT, XalanJ and XSLTC modes. Not allowed in the MSXML mode.
Nested comment	XML specification does not allow nested comments.	In the MSXML mode, comments such as <!--<!--abc-->--> do not cause an error.	Allowed in the MSXML mode. N/A in other modes.
Undefined key	Undefined in the specification.	• In the MSXML mode, using an undefined key does not cause an error. • In other modes, it returns an empty nodeset.	Allowed in the Standard, XT, XalanJ and XSLTC modes. Not allowed in the MSXML mode.

Appendix C: References

This section lists relevant external documents referenced in the current document.

1. W3C* XML 1.0 Recommendation, http://www.w3.org/TR/xml/
2. W3C* Namespaces in XML 1.0 Recommendation, http://www.w3.org/TR/2006/REC-xml-names-20060816/
3. W3C* XSLT 1.0 Recommendation, http://www.w3.org/TR/1999/REC-xslt-19991116.html
4. W3C* Document Object Model (DOM) Level 2 Core Specification, http://www.w3.org/TR/DOM-Level-2-Core/, Level 3 http://www.w3.org/TR/DOM-Level-3-Core/
5. Simple API for XML, http://www.saxproject.org/
6. XML W3C* Conformance Test Suite, http://www.w3.org/XML/Test/
7. W3C* XML Schema 1.0, http://www.w3.org/TR/xmlschema-1/, http://www.w3.org/TR/xmlschema-2/
8. XML W3C* XPath 1.0, http://www.w3.org/TR/1999/REC-xpath-19991116
9. Java* API for XML Processing (JAXP), https://jaxp.dev.java.net/
10. Apache Xalan* Java* transformer, http://xml.apache.org/xalan-j/
11. Apache Xalan* C transformer, http://xml.apache.org/xalan-c/
12. OASIS* XSLT Conformance test suite, http://www.oasis-open.org/committees/documents.php?wg_abbrev=xslt
13. Extension to XSLT, http://exslt.org/
14. Java* Signal Chaining, http://java.sun.com/j2se/1.5.0/docs/guide/vm/signal-chaining.html
15. http://wiki.eclipse.org/FAQ_What_is_the_classpath_of_a_plug-in%3F
16. http://wiki.eclipse.org/FAQ_How_do_I_add_a_library_to_the_classpath_of_a_plug-in%3F

Index

A

Apache* Xalan* tool 53

C

ContentHandler 29

D

Document Object Model 1

DocumentBuilderFactory 30

DOM (document object model) 1

E

ErrorHandler 42

EXSLT extensions 47

- Common 47
- Date-and-time 47
- declaring namespace 47
- Math 47
- Sets 47
- Strings 47
- using 50

extensible stylesheet language 1

external resources

- in schema validation 42
- in XPath 57
  - function resolver 58
  - namespace context 57
  - variable resolver 58

I

Intel® XML Parsing Accelerator 1

- using
  - in DOM mode 30
  - in SAX mode 29

Intel® XML Schema Accelerator 1

- samples 15
- using 40

Intel® XML Software Suite 1

- components of 1
- features 4

Intel® XPath Accelerator 1

- samples 17
- using 55

Intel® XSLT Accelerator 1

- samples 15
- using 44

J

JAXP (Java API for XML Processing) ... 4

P

Parsing See Intel® XML Parsing Accelerator, using 29

S

Samples 12

SAX (Simple API for XML) 1

SAXParser 29

SAXParserFactory 29

SchemaFactory..... 40

Simple API for XML See SAX.....1

T

Templates..... 44

Transformer 44

TransformerFactory..... 44

U

user-defined extensions..... 50

non-XSL input types 50

return values..... 50

using..... 50

XSL input types..... 50

V

validation of..... 40

Data..... 36

DTDs..... 38

validator See Intel® XML Schema Accelerator..... 40

X

XML Suite See Intel® XML Software Suite..... 1

XPath..... 55

XPathExpression..... 55

XPathFactory 55

XSL (extensible stylesheet language). 1

extensions 47

EXSLT See EXSLT extensions .. 47

other 53

user-defined See user-defined extensions..... 50

objects 46

DOM trees..... 46

SAX Events 46

streams 46

transformation..... 44

configuring 45

object types 46